Efficient Surface Reconstruction From Noisy Data Using Regularized Membrane Potentials

Jalba, Andrei C.; Roerdink, Jos B.T.M.

Published in:
Ieee transactions on image processing

DOI:
10.1109/TIP.2009.2016141

Publication date:
2009

Efficient Surface Reconstruction From Noisy Data Using Regularized Membrane Potentials

Andrei C. Jalba and Jos B. T. M. Roerdink, *Senior Member, IEEE*

***Abstract*—A physically motivated method for surface reconstruction is proposed that can recover smooth surfaces from noisy and sparse data sets. No orientation information is required. By a new technique based on regularized-membrane potentials the input sample points are aggregated, leading to improved noise tolerability and outlier removal, without sacrificing much with respect to detail (feature) recovery. After aggregating the sample points on a volumetric grid, a novel, iterative algorithm is used to classify grid points as exterior or interior to the surface. This algorithm relies on intrinsic properties of the smooth scalar field on the grid which emerges after the aggregation step. Second, a mesh-smoothing paradigm based on a mass-spring system is introduced. By enhancing this system with a bending-energy minimizing term we ensure that the final triangulated surface is smoother than piecewise linear. In terms of speed and flexibility, the method compares favorably with respect to previous approaches. Most parts of the method are implemented on modern graphics processing units (GPUs). Results in a wide variety of settings are presented, ranging from surface reconstruction on noise-free point clouds to grayscale image segmentation.**

***Index Terms*—Graphics processing units (GPU), mass-spring system, membrane potential, point cloud, regularization, surface reconstruction, volumetric segmentation.**

I. Introduction

In the area of surface reconstruction the purpose is to obtain a digital representation of a real, physical object or phenomenon described by a cloud of points, which are sampled on or near the object's surface. The growing interest in this field is due to the increasing availability of point-cloud data, such as may be obtained from medical scanners, laser scanners, vision techniques (e.g., range images), and other modalities.

In computer vision, shape recovery is a classical problem, whose goal is to derive a 3-D scene description (e.g., surface normal and surface depth) from one or more 2-D images. All techniques that recover shape are commonly called "shape-from-X," where X can be shading, stereo, texture, or silhouettes, etc. (see [1]–[6] and the references therein). For example, in the stereo problem, one first extracts features (e.g., corners, lines, etc.) from a collection of input images, and then solves the so-called correspondence problem, i.e., matching features across images. After obtaining depth information at the locations of the extracted features, one needs to reconstruct the surfaces of the objects present in the scene. One way of achieving this is by using techniques that reconstruct surfaces from point clouds.

Reconstructing a surface from unorganized point clouds is a challenging problem because the topology of the real surface can be very complex, the acquired data may be nonuniformly sampled and the data may be contaminated by noise. In addition, the quality and accuracy of the data sets strongly depend upon the acquisition methodology. Furthermore, the computational cost of reconstructing surfaces from large datasets can be prohibitive. Most of the existing reconstruction methods were developed postulating that precise and noise-free data is available. Therefore, they cannot meet the demands posed by noisy and/or sparse data.

In this paper, which is a greatly extended version of [7], we propose a novel, physically based technique for surface reconstruction, which employs regularized membrane potentials evaluated on a volumetric grid. Although the input data are noisy and sparse, the output surfaces are smooth. The purpose of the membrane potentials is twofold: first, to aggregate data points; second, to remove outliers due to noise. The process in which gaps between the data points are bridged by a slowly varying scalar field will be referred to as *aggregation*.

The contributions of this paper are as follows.

- A new method for aggregating input data points, based on regularized-membrane potentials, as an alternative approach to the widely employed distance transform (Section III-A). The main advantage of our surface-reconstruction approach is a greatly improved robustness with respect to noise.
- A fast, iterative algorithm for classifying grid points into exterior and interior to the surface (Section III-B).
- A new method for surface smoothing (Section III-C) based on a mass-spring system enhanced with a bending-energy minimizing term, ensuring that the final triangulated surface is smooth (i.e., smoother than piecewise linear).

Our formulation handles noisy as well as nonuniform data sets, it works in any dimension, and is very competitive with previous approaches as regards computing costs. We also take advantage of the increased computational power of modern graphics hardware, and present implementations of most parts of the method on graphics processing units (GPUs). The method can also be used to perform surface reconstruction starting from grayscale volumetric data, leading to *image segmentation*.

We demonstrate the flexibility and power of the proposed method in a wide variety of settings (Section V). In particular, we show that the proposed method: (i) quickly reconstructs surfaces of very large models, (ii) is reliable even under heavy (shot

Manuscript received June 10, 2008; revised January 21, 2009. Current version published April 10, 2009. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Xuelong Li.

The authors are with the Institute for Mathematics and Computing Science, University of Groningen, 9700 AK Groningen, The Netherlands (e-mail: andrei@cs.rug.nl; j.b.t.m.roerdink@rug.nl).

Digital Object Identifier 10.1109/TIP.2009.2016141

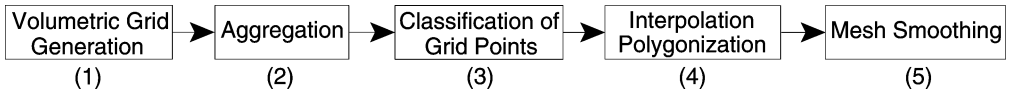

Fig. 1. Flow diagram of the proposed surface reconstruction method.

and Gaussian) noise conditions, and (iii) copes well with diverse inputs obtained from particle systems, contours, and grayscale volumetric data—all these in the absence of any orientation information.

II. Previous and Related Work

There are two main categories of surface representation, called explicit and implicit. We shall classify various existing techniques according to the kind of surface representation employed. As our method is based on an implicit representation, we shall focus our overview on this type. Two important representatives of the first class are parametric and triangulated surfaces. Parametric surfaces are represented by parametric patches (e.g., NURBS, B-spline and Bézier patches), which can represent smooth surfaces and can cope with nonuniform data. Major drawbacks are that patches should be combined to form closed surfaces (which can be very difficult for arbitrary data sets), and noise in the data set is difficult to handle. Triangulated surfaces are usually obtained using tools from computational geometry. Methods that rely on this representation extract subsets of faces of triangulations to yield the reconstructed surfaces [8]–[11]. These methods exactly interpolate the data, and, therefore, are rather sensitive to noise. Moreover, inserting hundreds of thousands of points into a triangulation is computationally expensive. Examples are Alpha Shapes [11], the (Power) Crust algorithm [8], [9], and the Ball-Pivoting algorithm [12].

When using implicit surface representations, one traditionally computes a signed distance function so that the reconstructed implicit surface can be represented by an iso-contour (usually at iso-value zero) of this function [13]–[17]. These methods require a way to distinguish between the inside and outside of closed surfaces. E.g., the method of Hoppe *et al.* [14] approximates the normal at each data point by fitting a tangent plane in its neighborhood, while Tang and Medioni [18] use the tensor-voting formalism to estimate the orientations. Both methods are sensitive to noise as they require accurate normal estimates. Zhao *et al.* [19] use the level-set formalism [20] for noise-free surface reconstruction. Their method can handle complicated topology and deformations, and the reconstructed surface is smoother than piecewise linear. The main drawback is the sensitivity of the method to shot noise, due to its reliance on the distance transform. While filtering the input 3-D data prior to reconstruction is certainly possible [21]–[23], our method does not need such preprocessing steps.

More recently, modeling of surfaces with Radial Basis Functions (RBFs) has become a popular technique [24]–[28]. Again, these methods are very noise-sensitive because local changes of the positions of the input points have global effects on the reconstructed surface. Morse *et al.* [25] and Ohtake *et al.* [29] use compactly supported RBFs to achieve local control and reduce the computational cost by solving a sparse linear system. Dinh *et al.* [27] use RBFs and volumetric regularization to handle noisy and sparse range data sets. Recently, Ohtake *et al.* [30] proposed a method based on the so-called "partition of the unity implicits," which can be regarded as the combination of algebraic patches and RBFs. Carr *et al.* [31] further address surface reconstruction from noisy range data by fitting a RBF to the data and convolving with a smoothing kernel during the evaluation of the RBF. Kojekine *et al.* [26] use compactly supported RBFs and an octree data structure leading to a band-diagonal system matrix, thus reducing computational costs. To conclude, the main advantages of implicit surface representations include topological flexibility, mesh independent and compact representation to within any desired precision. Moreover, efficient algorithms for polygonization of implicit surfaces are available [32]–[34].

III. Proposed Method

Fig. 1 shows the computational flow diagram of our method. First, the input sample points (assumed to be without any orientation information) are assigned to grid cells, using cloud-in-cell (CIC) interpolation [35] (first step in Fig. 1). Step 2 performs aggregation of the sample points by computing regularized-membrane potentials on the grid. A labeling algorithm, which follows increasing paths of the scalar field (starting from the bounding box and marching towards the data points) is used to classify the grid points into exterior and interior to the surface, thus defining an implicit (rough) surface (step 3). Prior to polygonization, we again use diffusion potentials, but this time with the purpose of producing a smooth implicit surface. Then, we employ Bloomenthal's polygonizer [34] to turn the implicit surface into a triangulated one (second part of step 4), and use a mass-spring system, enhanced with a bending-energy minimizing term, in order to obtain a larger degree of surface smoothness (step 5).

A. Aggregation of the Input Data Points

This step assigns the input data points to cells of a 3-D grid, using the CIC interpolation scheme. Accordingly, a constant numerical value (we fix this value to one), representing the contribution of each data point to the initial (heat) distribution, is spread to the eight nearest cell centers. The weights are given by the overlap volumes of a box, centered around the data point under consideration, with the neighboring voxels. If several points contribute to the same cell, the values are accumulated. As we will see below, the nonempty grid cells will serve as sources generating potentials on the grid.

The nonempty grid cells, called *source points*, are regarded as sources for the physical simulation of heat flow, as defined by the linear diffusion equation

$$\frac{\partial u}{\partial t} = \nabla^2 u \tag{1}$$

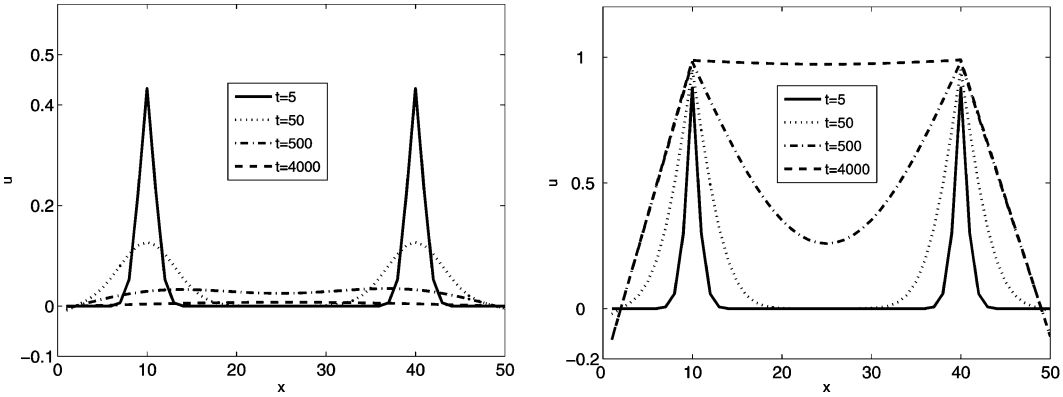

Fig. 2. Examples of 1-D diffusion at time steps $t = 5, 50, 500, 4000$; two sources with value 1 are placed at positions $x = 10$ and $x = 40$. *Left*: linear diffusion; *right*: regularized membrane equation [$\mu = 0.1$, see (3)].

where u is the concentration of diffusing material, with the original volume data f as initial condition, i.e., $u(t = 0) = f$. Aggregation using (1) has the disadvantage that it converges to a constant steady state. That is, the sizes of the support regions around the cells corresponding to the input points increase with the number of iterations, so that the diffusion would eventually converge to a constant solution covering the whole volume. An illustration of this effect, in the 1-D case, is shown in the left graph of Fig. 2. Two heat sources are placed at positions $x = 10$ and $x = 40$. From the temporal evaluation of the pure diffusion process, one can easily notice that after $t > 500$ the positions of the two maxima (corresponding to the sources) can barely be distinguished.

Since we are not interested in the steady state of linear diffusion, a criterion is required for choosing a stopping time. This can be done with the help of an additional reaction term, which keeps the steady state close to the initial value, leading to the *regularized membrane equation* (see also [36]–[38])

$$\frac{\partial u}{\partial t} = \nabla^2 u + \beta(f - u). \tag{2}$$

The physical analogy of (2), without the diffusive term, is a patch of passive neural membrane, which can be modeled with a serial RC (resistor, capacitor) circuit. The capacitor represents the fact that cellular membranes are good electrical insulators, whereas the resistor depicts the leakage of current through the membrane. The "β" term in (2) demarcates the amount by which the current in the RC circuit has diverged from its original value. This term ensures that the reaction-diffusion equation reaches a steady state not far from the original values of f. However, the problem of choosing a proper stopping time for the linear diffusion is shifted to finding a suitable value for the parameter β. To alleviate this problem we have chosen the value of β equal to the absolute value of the original signal f at each voxel location x, yielding

$$\frac{\partial u}{\partial t} = \mu\nabla^2 u + |f|(f - u) \tag{3}$$

where μ is a small regularization constant which controls the amount of smoothing. Note that we have used $\beta = |f|$ in (3) so that we can also handle negative values of f; this will be necessary when performing interpolation in Section III-C, see (4). The choice of μ is not critical, as we show in Section V.

Although a closed-form solution of (3) is not at all trivial to obtain, one can use the method of eigenfunction expansion to find an approximate solution. The temporal behavior of the solution using the first 20 terms in the eigenfunction expansion is illustrated in the right graph of Fig. 2. Note that the positions of the maxima can easily be demarcated even for large values of the parameter t, i.e., only after $t > 4000$ a plateau of constant value appears between the two maxima. In fact, the steady-state solution of this equation linearly interpolates the data points, whereas the transient solutions are equivalent to Gaussian interpolants in space which decay exponentially in time. Therefore, one can conclude that the formulation of the diffusion process given in (3) is suitable for the purpose of aggregating the input points, provided that the process is stopped before reaching some huge value of the time parameter t. However, the diffusion process should not be stopped too early, because then the "gaps" between the projected input points will not be bridged.

B. Classification of Grid Cells

After aggregation, a method is needed which separates the exterior grid points from the interior ones, thus defining the primary implicit surface. This method should start from the bounding box of the computational grid, follow increasing paths of the scalar field on the grid towards the source points, and label grid cells as exterior, as it proceeds. After the propagation has stopped at regional maxima and ridges, the boundary separating the remaining (interior) points from the exterior points can be traced to yield the reconstructed surface. For this purpose, we will first examine the tagging algorithm of Zhao *et al.* [19] and then propose a fast, iterative algorithm that fulfills our requirements.

The tagging algorithm of Zhao *et al.* [19] starts by labeling points on the bounding box of the computational domain as exterior and all other points as interior. This method assumes that the (un-signed) distance transform of the grid cells corresponding to the input points has been computed on the grid. Then, those interior points that have at least one exterior neighbor are labeled as temporary (unknown) boundary points and are inserted into a *sorted heap*. Next, the subset defined by

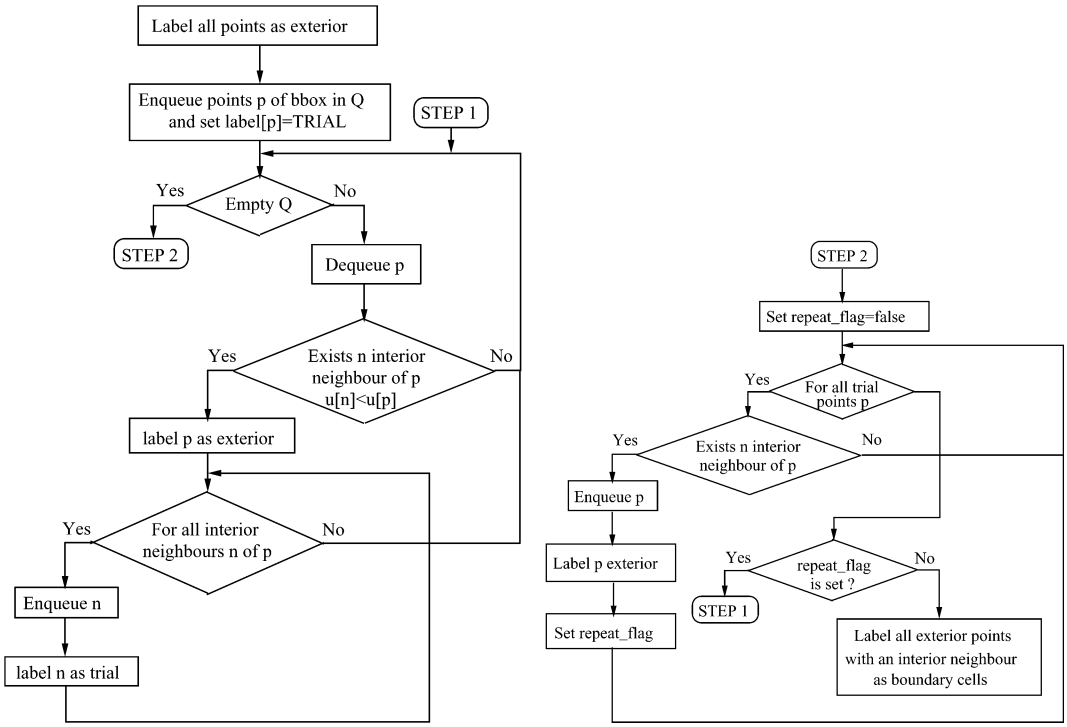

Fig. 3. Flowchart diagram of our labeling algorithm.

the remaining interior points is swept to march the temporary boundary points inwards, towards the input points, as follows. The temporary boundary point with the largest distance (on top of the heap) is checked whether it has at least one interior neighbor with a larger or equal distance value. If not, the point is taken out from the heap, turned into an exterior point, and all its interior neighbors are inserted into the heap. If it does have such a neighbor, the point is removed and turned into a final boundary point (none of its neighbors is added to the heap). This process is repeated until the maximum distance of the temporary boundary points is smaller than some preset distance threshold.

Algorithm 1 Labeling of Grid Points as Exterior, Boundary, and Interior

1: Label all points p as INTERIOR (assign label[p]:=INTERIOR)
2: Enqueue points p of bounding box in queue Q and assign label[p]:=TRIAL
3: **repeat**
4: **while** Not empty queue Q **do**
5: p:=Dequeue Q
6: **if** $\nexists$n neighbor of p with label[n]=INTERIOR and u[n]+ε< u[p] **then**
7: label[p]:=EXTERIOR
8: **for** all neighbors n of p with label[n]=INTERIOR **do**
9: Enqueue n
10: label[n]:=TRIAL
11: repeat_flag:=false
12: **for** all points p with label[p]=TRIAL **do**
13: **if** $\exists$n neighbor of p with label[n]=INTERIOR **then**
14: Enqueue p
15: label[p]:=EXTERIOR;
16: repeat_flag:=true
17: **until** repeat_flag=false
18: **for** all points p with label[p]=EXTERIOR **do**
19: **if** $\exists$n neighbor of p with label[n]=INTERIOR **then**
20: label[p]:=BOUNDARY;

Since each grid point is visited at most once, the tagging algorithm described above has a time-complexity of $O(M \log M)$, where M is the number of grid points, and the factor $\log M$ comes from sorting. This algorithm is computationally expensive for large values of M. Therefore, instead of adapting this algorithm to meet our requirements, we have developed a fast iterative sweeping algorithm, the pseudo-code of which is given in Algorithm 1 (see also Fig. 3 for its flowchart diagram).

Our algorithm starts also by labeling all points as interior (line 1). Then, the points situated on the bounding box of the grid are inserted in a *queue* (enqueued) and assigned some temporary value, TRIAL. Then, the subspace is swept as follows. Each trial point is removed from the queue (dequeued) and checked to see if it has at least an interior neighbor that has a smaller value of the potential scalar field (line 6). Only if it does not have such a neighbor, the point is turned into an exterior point and all its interior neighbors are inserted into the queue, as trial points. Otherwise, none of its neighbors is enqueued and its label remains untouched. This case, in which the marching front reaches an interior point with a smaller value, may occur in two situations: (i) either the front has just arrived at the true location of the boundary separating surface interior from exterior, or (ii) the point has a neighbor which has been labeled beforehand. In the first case, the algorithm should stop turning inte-

rior neighboring points into exterior points, since these points are situated on the other side of the advancing front, and they are truly interior points. The second case is usually encountered when the marching front approaches concave regions, and it happens because the algorithm uses a queue as opposed to a sorted heap. However, we can solve this ambiguity in the following iterative manner. All trial points which have at least an interior neighbor are deemed (temporarily) exterior points and enqueued. Also, the existence of such points is signaled by turning the boolean variable `repeat_flag` to `true` (lines 12–16). Then, the whole process is repeated within the **repeat-until** loop, until no more such points remain. At the end of the algorithm (lines 18–20), those exterior points which have at least one interior neighbor are labeled as boundary points. Since each point is visited at most twice, one iteration of the algorithm will be completed in no more than $O(M)$ operations. However, a concern may arise regarding the number of iterations (in the **repeat-until** loop) required by the algorithm to converge. Although this number is data-dependent, we did not encounter in our experiments any data set for which the algorithm would not terminate in less than five iterations. In practice, our algorithm is faster than the tagging algorithm of Zhao *et al.*, while CPU timings suggest a linear-time algorithm. The classification of grid cells can also be formulated as a linear convection problem, whose velocity is the gradient of the membrane potential, unlike Zhao *et al.* who use the gradient of the distance function. Further, one can simplify the convection problem and turn it into an eikonal one, which can be solved in linear time using the fast sweeping method of [39]. Although we did not explore this possibility, we believe that our fast iterative algorithm is better suited for grid labeling as it does not need to evaluate the gradient of the membrane potential, which is singular at the positions of the nonempty grid cells.

C. Surface Smoothing and Polygonization

After classification, one can use Bloomenthal's method [34] to polygonize the implicit surface given by the zero level set of the scalar field g defined by

$$g(x,y,z) = \begin{cases} -1, & \text{if } (\mathrm{x},\mathrm{y},\mathrm{z}) \text{ is labeled as } \texttt{INTERIOR} \\ 0, & \text{if } (\mathrm{x},\mathrm{y},\mathrm{z}) \text{ is labeled as } \texttt{BOUNDARY} \\ 1, & \text{if } (\mathrm{x},\mathrm{y},\mathrm{z}) \text{ is labeled as } \texttt{EXTERIOR}. \end{cases} \tag{4}$$

1) Interpolation Using Membrane Potentials: Direct polygonization will cause "staircase" artefacts in the resulting mesh (see Fig. 4, left column). A better approach is to interpolate the implicit surface using the reaction-diffusion process (3) a second time, with the labeled grid points as sources. Sources are instantiated only at the locations of the interior and exterior grid points [see (4)], since the membership of boundary points is uncertain. By tracing the zero iso-contour a smooth scalar field emerges and the implicit surface is turned into a triangulated one. Since boundary voxels form thin bands along surface borders, a small number of iterations is required, resulting in fast computation. The triangulated surface, which is a better approximation to the real surface than the initial one, is used as initialization for the more computationally demanding mass-spring system, described next.

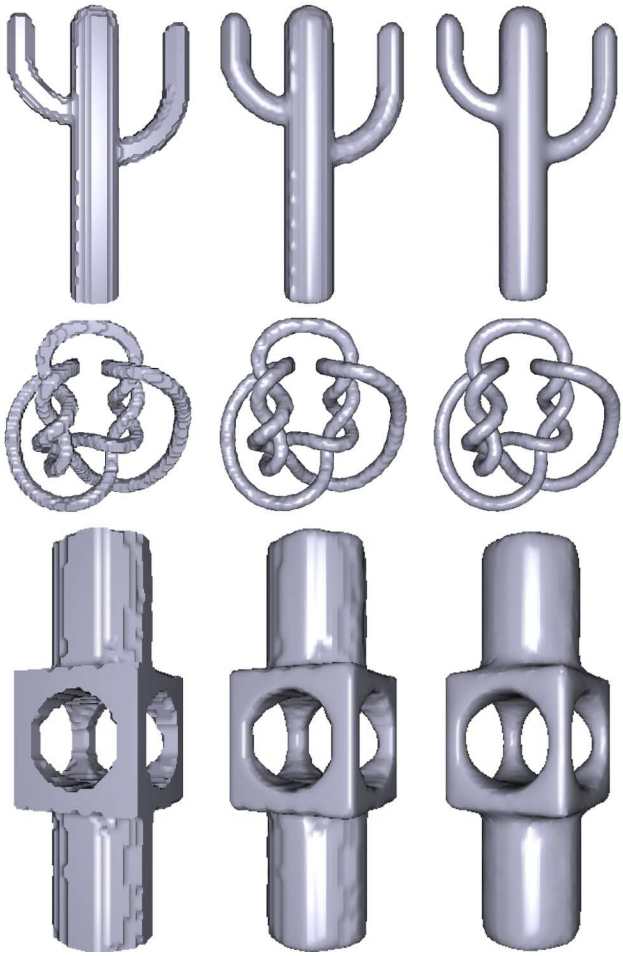

Fig. 4. Smoothing examples (grid resolutions: $33 \times 33 \times 80$, $79 \times 80 \times 43$, $21 \times 54 \times 80$). *Left*: no smoothing; *center*: smoothing by reaction-diffusion potentials; *right*: smoothing by the reaction-diffusion process and the enhanced mass-spring system.

2) Mesh Smoothing With a Mass-Spring System: Assuming that the correct topology has been inferred and the triangulated surface possesses consistent orientation (see Section V-D for a justification), we propose a mass-spring system for obtaining a larger degree of smoothing. That is, each edge of each triangular patch comprising the mesh is modeled by a spring and each vertex is regarded as a particle with a small mass. Since we utilize triangular elements, we do not need to include extra cross springs to afford resistance against shearing (see [40]). In addition, we integrate an extra energy term such that the *bending energy* of the system is minimized. This has the beneficial effect, analogous to curvature flow, that the triangulated surface is smoothened by moving its vertices along their normals with a speed proportional to the (normal) curvature.

We start by defining nodes p_i, $i = 1, 2, \ldots, N$, of the mass-spring network, where node p_i has mass m_i and position vector $\mathbf{x}_i(t) = [x_i(t), y_i(t), z_i(t)]$. We denote by $\mathcal{N}_i$ the set of neighbors of p_i, i.e., all particles p_j with an edge e_{ij} between p_i and p_j. Let spring s_{ij} connect nodes p_i and p_j, have rest length l_{ij} and stiffness $c_{ij} = c$, where c is a constant; also, let $\mathbf{r}_{ij} =$

$\mathbf{x}_j - \mathbf{x}_i$ be the vector separating the two nodes. Then the energy of spring s_{ij} can be expressed as

$$E_{s_{ij}} = \frac{c}{2}(|\mathbf{r}_{ij}| - l_{ij})^2. \tag{5}$$

The potential energy $E_{s_{ij}}$ of the spring s_{ij} gives rise to a force $\mathbf{f}_{s_{ij}}$ acting on particle p_i due to particle p_j

$$\mathbf{f}_{s_{ij}} = -\nabla_{\mathbf{x}_i}(E_{s_{ij}}) = c(|\mathbf{r}_{ij}| - l_{ij})\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|}. \tag{6}$$

Smoothing a mesh by minimizing a membrane energy functional [37] can be regarded as the physical simulation of a mass-spring network with zero-rest length springs that will shrink to a single point. On the one hand, because such behavior of the mass-spring system is undesirable for our purposes, the rest lengths of the springs should be chosen such that they reflect the lengths of the edges of the initial (un-deformed) mesh. On the other hand, in order to facilitate the relaxation of the mesh structure into a smooth configuration, the rest lengths of the springs should be smaller than the initial lengths of the edges of the mesh, i.e., we use a percentage of the initial edge lengths.

The bending energy of an ideal, thin flexible flat plate of elastic material is defined as the sum of squared curvatures along the surface. We modify this definition of bending energy slightly, to restrict it to the neighborhood of a particle p_i as

$$E_{b_i} \equiv \sum_{j\in\mathcal{N}_i} E_{b_{ij}} = \frac{1}{2}\sum_{j\in\mathcal{N}_i} k_{ij}^2 \tag{7}$$

where k_{ij} is some discrete curvature measure between the particle pair (p_i, p_j). A (mean) curvature estimate, which has been previously used in the context of mesh smoothing [41], [42], is given by $k_{ij} = 2\,(\mathbf{n}_i \cdot \mathbf{r}_{ij})/|\mathbf{r}_{ij}|^2$.

Since $E_{b_{ij}} = k_{ij}^2$, the conservative force $\mathbf{f}_{b_{ij}}$ acting on particle p_i due to particle p_j, minimizing the bending energy, is

$$\mathbf{f}_{b_{ij}} = -\nabla_{\mathbf{x}_i} E_{b_{ij}} = \frac{4(\mathbf{n}_i \cdot \mathbf{r}_{ij})\mathbf{n}_i}{|\mathbf{r}_{ij}|^4} - \frac{8(\mathbf{n}_i \cdot \mathbf{r}_{ij})^2\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^6}. \tag{8}$$

Thus, according to (8) the movement of particle p_i due to its interaction with particle p_j will be primarily in the direction of $\mathbf{n}_i$, whereas the second term in (8) forces the particle to slightly move in the direction of $-\mathbf{r}_{ij}$, to compensate for the former movement. The normals $\mathbf{n}_i$ are computed as the average of the normals of the triangular faces incident to each point p_i.

The potential energy of a particle p_i of the mass-spring system due to its interactions with a neighboring particle p_j, $j \in \mathcal{N}_i$ is given by

$$E_i = \alpha E_{s_{ij}}(\mathbf{r}_{ij}) + (1-\alpha)E_{b_{ij}}(\mathbf{r}_{ij}, \mathbf{n}_i) \tag{9}$$

where the first term represents the energy (5) of the spring connecting the particles, the second term is the bending energy (7), and α is a scalar weight.

The variations of particle potentials with respect to positions yield forces acting on particles. The corresponding system of differential equations is

$$\mathbf{f}_i = \sum_{j\in\mathcal{N}_i} (\alpha \mathbf{f}_{s_{ij}} + (1-\alpha)\mathbf{f}_{b_{ij}}) \tag{10}$$

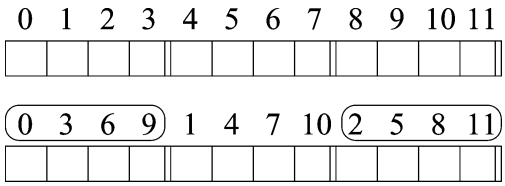

Fig. 5. Packing a row of data into an RGBA texture. The numbers indicate indexes before (*top*) and after packing (*bottom*).

$$\begin{cases} \frac{d\mathbf{x}_i(t)}{dt} = \mathbf{v}_i \\ \frac{d\mathbf{v}_i(t)}{dt} = \mathbf{a}_i(t) = -\nabla_{\mathbf{x}_i} E_i = \mathbf{f}_i(\mathbf{x}_i(t), \mathbf{n}_i(t)) \end{cases} \tag{11}$$

where the potential energy E_i is given by (9), and $\mathbf{x}_i$, $\mathbf{v}_i$ and $\mathbf{a}_i$ are its position, velocity and acceleration, respectively (the mass has been set to one). The problem with integrating (11) is the circular dependency between $\mathbf{x}_i(t)$ and $\mathbf{f}_i(\mathbf{x}_i(t), \mathbf{n}_i(t))$, which restricts the methods for numerical integration which can be used. A "velocity-less" integration method, widely used in molecular-dynamics simulations, is the so-called Verlet method [43]. Instead of storing the position and velocity of each particle p_i, this method stores its current position $\mathbf{x}_i$ and its previous position $\mathbf{x}_i^*$. Then, the implicit update rule for the new position is

$$\mathbf{x}_i' = \mathbf{x}_i + \mathbf{v}_i' \mathrm{d}t = \mathbf{x}_i + (\mathbf{v}_i + \mathbf{a}_i \mathrm{d}t)\,\mathrm{d}t. \tag{12}$$

Since the current velocity $\mathbf{v}_i$ can be approximated as $(\mathbf{x}_i - \mathbf{x}_i^*)/\mathrm{d}t$, the integration step becomes

$$\begin{cases} \mathbf{x}_i' = \mathbf{x}_i + (\mathbf{x}_i - \mathbf{x}_i^*) + \mathbf{a}_i \mathrm{d}t^2 \\ \mathbf{x}_i^* = \mathbf{x}_i. \end{cases} \tag{13}$$

Accounting for a small amount of drag (damping), the update rule becomes

$$\mathbf{x}_i' = \mathbf{x}_i + (1 - \gamma \mathrm{d}t)(\mathbf{x}_i - \mathbf{x}_i^*) + \mathbf{a}_i \mathrm{d}t^2. \tag{14}$$

The main advantage of the Verlet method is that it is fast and stable since the velocity is handled implicitly. Yet, it is not always very accurate, since energy may dissipate from the system.

D. Mesh Smoothing Examples

Experiments were performed at *small grid resolutions*, such that the effect of smoothing is emphasized. The first column of Fig. 4 shows triangulated surfaces without smoothing, i.e., we simply traced the zero iso-contour after labeling the grid points. Note that in this case, all reconstructed surfaces are blocky, though one should consider that we specifically used very small computational grids. The second column shows the results when the reaction-diffusion process was employed prior to polygonization. As expected, the resulting surfaces are smoother, but this degree of smoothing usually does not suffice. The last column of Fig. 4 shows smooth triangulated surfaces obtained after applying the mass-spring system and stopping its evolution at $t = 5$ (corresponding to 50 Verlet iterations with time step $\mathrm{d}t = 0.1$).

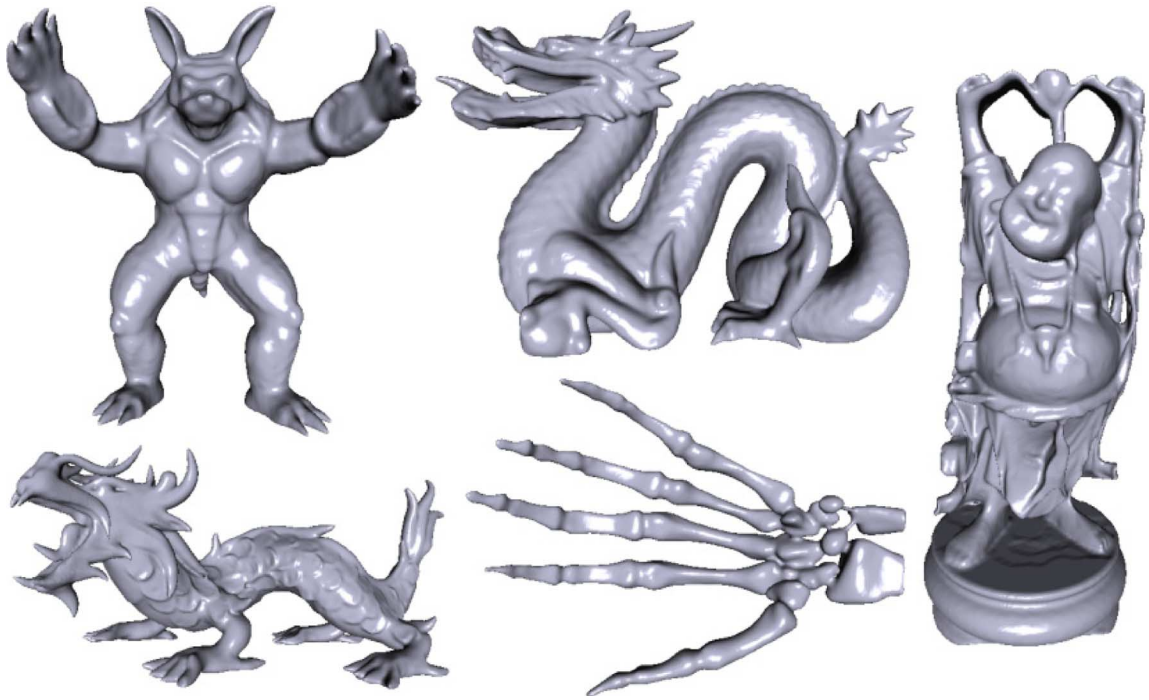

Fig. 6. Reconstruction of large models, see Table I.

IV. Implementation on GPU Hardware

In this section, we describe GPU implementations of most constituent parts of the proposed surface-reconstruction method: (i) an iterative method for the computation of the regularized-membrane equation, (3), and (ii) an implementation of the mass-spring system from Section III-C2. Referring to Fig. 1, this means that the aggregation stage [step (2)] and both smoothing methods, based on nonlinear diffusion and on the mass-spring system [partly step (4) and step (5)] are implemented on GPU hardware.

A. Regularized-Membrane Potentials

The regularized-membrane equation, (3), is discretized using finite differences (forward differences in time and central differences in space), such that the discrete update rule is

$$u_{i,j,k}^{n+1} \leftarrow u_{i,j,k}^{n} + \Delta t \left[\mu \sum_{(l,m,n)\in\mathcal{N}_6} u_{i+l,j+m,k+n}^{n} - 6u_{i,j,k}^{n} + |f_{i,j,k}|(f_{i,j,k} - u_{i,j,k}^{n}) \right]$$

where Δt is the time step ($\Delta t \leq (\Delta x \Delta y \Delta z)/(6\mu)$ for stability reasons), n is the current iteration, and $\mathcal{N}_6$ denotes the neighbors of location (i,j,k) using 6-connectivity. This update rule can be straightforwardly encoded in a fragment program running on GPU hardware, as follows. Volumes f and u (initially $u(t=0)=f$) are encoded as luminance (or red color) components into two collections T_f and T_u of 2-D texture maps. At each iteration, each texture $t_u \in T_u$ representing a slice of the 3-D volume u is updated by (i) enabling the fragment program, (ii) passing it textures t_f, t_u, left (t_u^-) and right (t_u^+) neighboring textures of t_u, and parameters Δt, μ, and (iii) rendering a quad which triggers the computation. At the end of the computation, temporary texture t_{t1} contains the updated values. Note that to update the next slice t_u^+, the previous values of t_u are needed, and not the updated values stored in t_{t1}. After slice t_u^+ has been updated as well, we store its values into a second temporary texture t_{t2}. Then, texture t_{t1} is copied at its proper location (i.e., t_u), and the temporary textures are swapped. The computations for the next slices proceed in a similar manner. Special care should be taken when updating the first and last slices, such that the desired (Dirichlet) boundary conditions are implemented.

A problem with this approach lies in the encoding of volumes u and f as scalar-only components into textures. Texture storage allows to represent vectors of up to four components, corresponding to the RGBA color components. This approach will not only be more efficient in storage but also in computations, since four scalar operations can be performed in parallel by GPUs, at no additional computational cost. However, if we simply encode 4-component vectors of data into RGBA textures along rows, the neighbors which would be retrieved by performing texture lookups at the left and right of the current position would not be the correct ones. An example is shown in the first row of Fig. 5. Assume that the neighbors at the left of the location in the middle (starting with index 4) are needed. By performing the lookup, the values with indexes (0,1,2,3) are retrieved. However, these are not the correct ones, e.g., the value with index 0 is not a left neighbor (in the original data) of the value with index 4, etc. To address this, we pack the data along rows using the permutation $(4j+i) \leftrightarrow (i(W)/(4)+j)$, where W is the width of the texture, $i=0,1,2,3$ and $j=0,1,\ldots,(W)/(4)$. For the case in Fig. 5, the result after packing is shown in the second row. Now the desired operation can be performed in parallel for four values. However, a new issue appears: the values in the first and last locations of any row (marked in Fig. 5) do not have left and right

TABLE I
STATISTICS, RECONSTRUCTION QUALITY, AND TIMINGS OBTAINED USING THE MEMBRANE POTENTIAL FOR AGGREGATION

Model	No. Points	Grid	No. Vertices	No. Triangles	Error (%)	Aggregation Time (s)	Marching Time (s)	Interpolation Time (s)	Smoothing Time (s)	Total Time (s)
Buddha	543,625	170x400x170	297,062	594,182	**0.08**	1.3	6.9	0.9	4.9	**15.2**
Armadillo	172,974	337x400x307	370,840	741,572	**0.05**	4.9	29.3	3.8	5.4	**41.8**
Dragon	433,375	400x284x184	383,680	767,390	**0.08**	2.1	13.4	1.5	6.9	**24.6**
Hand	327,323	400x283x143	190,664	381,372	**0.06**	1.6	11.7	1.2	3.1	**18.2**
Asian Dragon	3,609,600	400x226x269	206,142	412,290	**0.06**	2.5	18.7	1.8	4.3	**28.2**

neighboring locations, respectively, although in the original data they do. Still, the corresponding values can be retrieved by performing lookups at the very last position for the left neighbors of the first locations, and conversely, at the first position for the right neighbors of the last locations. Of course, there exists no left neighbor of location 0 nor a right one for the last location (11 in our example). The unpacking scheme needed to retrieve the values when the whole computation has been finished can easily be deduced.

In our implementation, first the "central region" of each slice is updated by rendering a quad with position $(1, 1, W-1, H-1)$. Then, two horizontal lines with y coordinates 0 and H, and two vertical lines with x coordinates 0 and W are rendered, which are required anyway to implement boundary conditions. The fragment programs for rendering the vertical lines also take care of the lookups specific to our packing scheme. The advantage is that no extra texture lookups into dependent textures are needed, to implement the indirect texture addressing otherwise required, see [44].

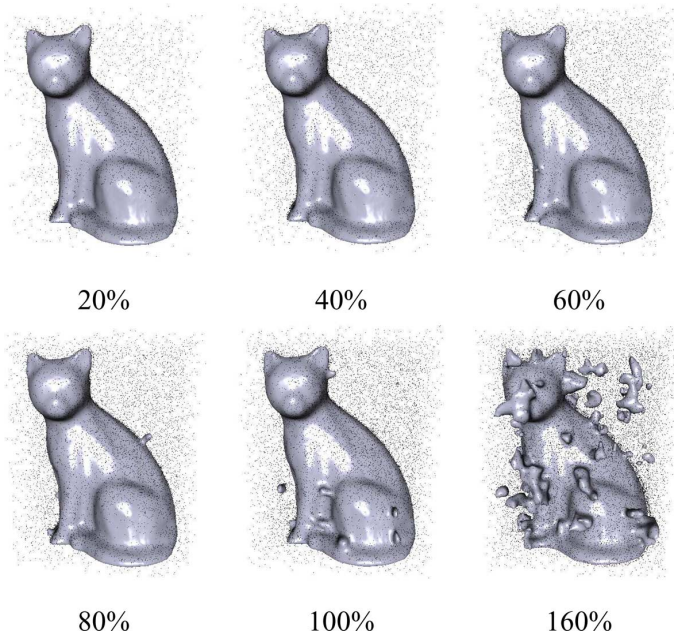

Fig. 7. Shot noise. The numbers indicate the fraction of corrupted voxels expressed as percentage of the number of source points.

B. Enhanced Mass-Spring System

The main computational steps required to update the position x_i of a particle p_i are (i) estimation of surface normal n_i, (ii) computation of the resulting force f_i on the particle (see (11)), and (iii) position update using the update rule in (13).

Normals $\boldsymbol{n}_i$ are computed as the average of the normals of the triangular faces incident to each vertex p_i. Thus, for each vertex p_i one needs to maintain a list of triangular faces incident to p_i. Likewise, a list of vertices adjacent to vertex p_i is needed when computing the spring force acting on p_i. Both lists are computed and stored into textures using a layout similar to that in [45]. The details are as follows.

Particle positions and vertex indexes of each face are encoded in two RGB textures t_x and t_f, respectively. For storing surface normals we use two additional textures $t_{N,\mathrm{addr}}$ and $t_{N,\mathrm{faces}}$, initialized as follows. For each vertex p_i, the indexes of the incident faces in p_i are stored consecutively into the RG components of texture $t_{N,\mathrm{faces}}$ as segments (i.e., along rows). Each segment's starting address and length are stored in the indirection texture $t_{N,\mathrm{addr}}$. The computations are triggered by drawing a quad with size equal to that of texture t_x, and they are performed by a fragment program which is passed textures t_x, t_f, $t_{N,\mathrm{addr}}$, $t_{N,\mathrm{faces}}$ as input parameters. First, the fragment program performs a lookup into texture $t_{N,\mathrm{addr}}$ to obtain the address of its corresponding segment from texture $t_{N,\mathrm{faces}}$. Then, each incident face is identified by performing a lookup into texture $t_{N,\mathrm{faces}}$ along the current segment. Next, the indexes of the vertices of the current incident face are found from texture t_f. Finally, the positions of the vertices of the current incident face are retrieved from texture t_x, and the contribution of this face to the normal in the current vertex p_i (corresponding to the current fragment) can be computed. At the end of the computation, the normals are stored in an OpenGL vertex array V_a.

An approach similar to the one just described, which uses two additional textures $t_{X,\mathrm{addr}}$ (for indirection encoding) and $t_{X,\mathrm{neigh}}$ (for adjacency), is used for the computation of the resulting forces as given in (11). However, it is computationally cheaper to combine this step with the last step, i.e., position update. According to (13), at the end of each integration step the current position x_i of particle p_i is stored in variable x_i^*. Thus, a useful optimization is to simply swap array pointers (in

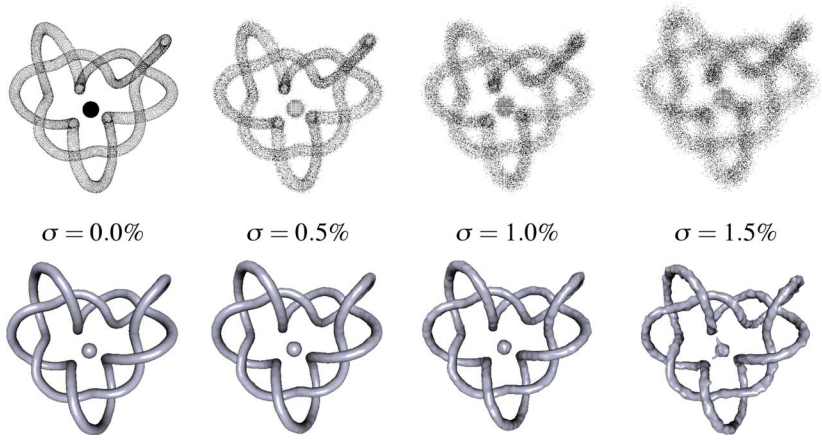

Fig. 8. Gaussian noise (zero mean). The numbers indicate the standard deviation (expressed as percentage of the length of the diagonal of the bound box of the grid) used to perturb the source points; *first row*: source points, *second row*: reconstructed surfaces.

terms of GPU hardware this means exchanging textures). We use an extra texture t_{x^*} which stores particle positions from the previous time step. The fragment program computes the force acting on the current particle (associated with the present fragment), then uses (13) to update its position. At the end of the computations, the new positions are made available into texture t_x and vertex array V_a. (The vertex array can be used, if desired, to render the mesh during its evolution.) When advancing to the next step, texture t_x is used for computing the normals, and then the two textures t_x and t_{x^*} are swapped.

The approach outlined above is implemented in two rendering passes: one for the computation of normals and the other for updating particle positions. However, the size of textures $t_{N,\text{faces}}$ or $t_{X,\text{neigh}}$ may become larger than the maximum texture size allowed by OpenGL. In this case, the surpluses are encoded into additional textures and extra rendering passes are needed to finalize the whole computation. Actually, very large data sets of hundreds of thousands of vertices (see Section V) require four passes, two for each computational step (computation of normals and update of particle positions).

The speedups obtained on a GeForce FX 7900 GTX graphics processor compared to optimized CPU algorithms (run on a machine with an AMD Opteron processor at 2.4 GHz) vary from 7 to 10 for the computation of the regularized-membrane potentials. Also, the mass-spring system performs at interactive rates of about 10 to 25 fps, for fairly large meshes, see Section V. A possible optimization is to use an algorithm for mesh decimation prior to mesh smoothing, so that the number of vertices and faces becomes smaller.

Currently, the grid point classification and polygonization steps are performed on the CPU, see Fig. 1. The reason is that current GPU hardware does not permit an efficient implementation of a flooding process, analogous to a standard CPU counterpart using a queue. Of course, it is possible to achieve the same result using an iterative brute-force method. Yet, the performance will not be any better than that of a sparse, queue-based computation run on the CPU. Although first attempts have been made to surface polygonization on GPUs [46], the performance was worse than that of standard CPU implementations. Moreover, in our method this step is performed only once, instead of many times as required for other applications (e.g., iso-surface browsing).

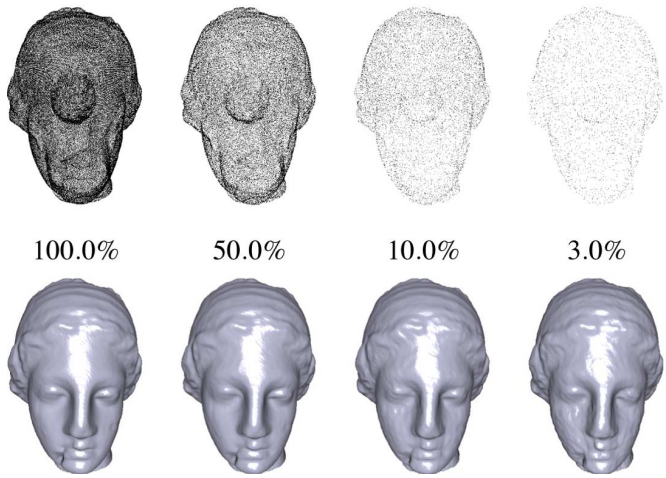

Fig. 9. Random sampling. The numbers indicate the fraction of sampled points, expressed as percentage of the number of source points; *first row*: sampled points, *second row*: reconstructed surfaces.

V. Results

All computations were performed on a system equipped with a Pentium IV processor at 3.0 GHz and a GeForce FX 7900 GTX GPU.

A. Large Data Sets

The parameters were set as follows. Aggregation was done by computing the membrane potential, as in Section IV-A; we used $N_m = 20$ iterations and the value of parameter μ was set to 0.1. The parameters of the Verlet integrator [see (14)] for the mass-spring system were $\mathrm{dt} = 0.1$ and $t = 10$. The force weight in (10) was set to $\alpha = 0.1$, to emphasize the bending-energy minimizing term. The rest lengths of the springs were set to 90% of the initial edge lengths. Finally, the largest dimension of the grid was set to 400 and the remaining two dimensions were obtained by uniform scaling of the bounding box of the

Fig. 10. Mesh smoothing comparison. *Left-to-right, top-to-bottom*: original mesh, method of Jones *et al*. [49], curvature flow [47], [48], our method.

sample points. Below we use the same values of the parameters (unless stated otherwise).

The results are shown in Fig. 6. Timings (in seconds) of each step of the method, for the models shown in Fig. 6, are given in Table I. The time taken is well under 1 min. The most expensive parts are the labeling and the second stage of smoothing by the enhanced mass-spring system. The computations have the same order of magnitude, and the computational cost generally depends exclusively on the grid size (e.g., compare the Asian Dragon model with the Armadillo model).

Table I provides some statistics. The sixth column shows the approximation error—an indication of the quality of reconstruction. This error is an upper bound for the average distance from the data points to the surface, and it is computed as the average distance from the data points to the centers of mass of the mesh triangles. The error is given as a percentage of the diagonal of the bounding box of the data points.

B. Coping With Noise

Next we consider the method under noise and random subsampling conditions.

1) Shot Noise: A certain number of empty voxels was changed by assigning them the value one, i.e., the same numeric value used to assign the input points. The number of corrupted voxels is expressed as a percentage of the number of *source points*. We used nearest-neighbor interpolation for grid assignment, as this results in a binary volume and represents a fair setting, without *a priori* information. The results of this experiment are shown in Fig. 7; the initial number of source points was 9 830.

For the computation of the membrane potential, the number of iterations N_m was increased from 20 to 100. The reason is that a large number of iterations results in a large aggregation support covering most of the exterior volume around the object, which will be correctly labeled as exterior. Note that the method is able to reconstruct the surface of the cactus shown throughout Fig. 7 even when as much as 80% of the source points were corrupted by noise.

2) Gaussian Noise: The input points were corrupted with zero mean Gaussian noise with standard deviations $\sigma = 0.5, 1.0, 1.5(\%)$, expressed as percentages of the length of the diagonal of the bound box of the grid. The results are shown in Fig. 8. The grid size was $210 \times 200 \times 114$. The parameters were set as in the previous section, with one exception—the stopping time t of the mass-spring system was increased from 10 to 25.

Unlike methods which rely on distance transforms, our method can cope with large amounts of Gaussian noise. In fact, in the third case ($\sigma = 1.0\%$) of Fig. 8, one percent of the diagonal of the bounding box means that $\sigma = 3.1$, which implies that the coordinates of most points were randomly translated in the interval $[-9.3; 9.3]$. Yet, even then the method is able to output smooth surfaces, with errors bounded by ε_r.

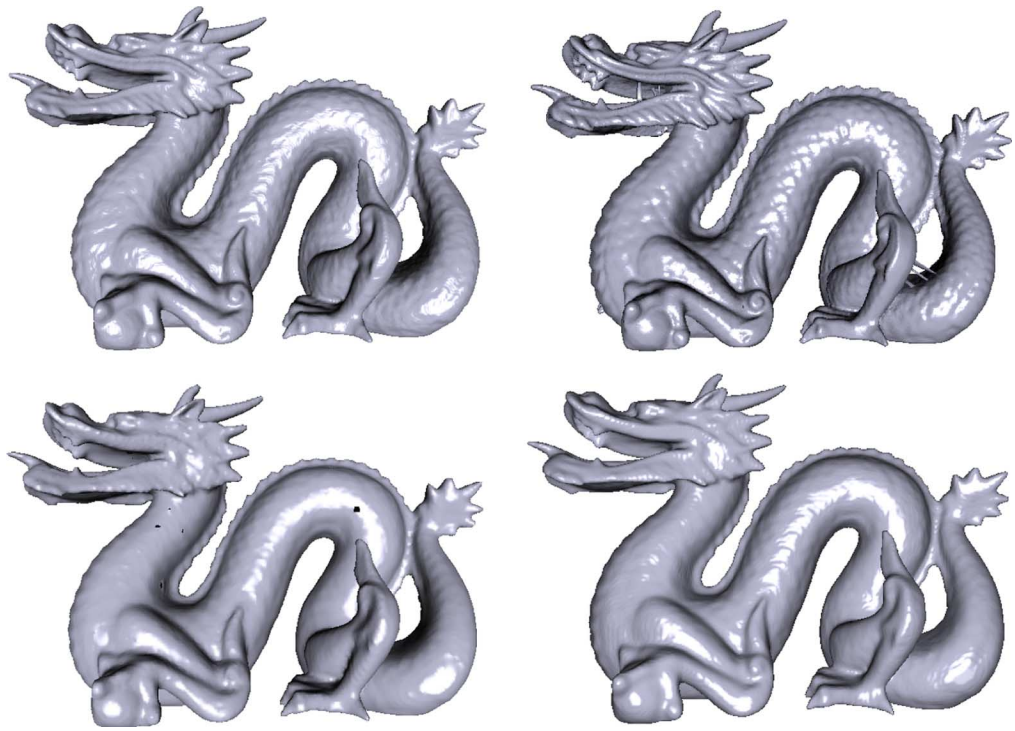

Fig. 11. Surface reconstruction comparison. *Top-left*: reconstruction by the proposed method; *Top-right*: Power Crust algorithm [8], [9]; *Bottom-left*: the method of Hoppe *et al*. [14]; *Bottom-right*: level set method of Zhao *et al*. [19].

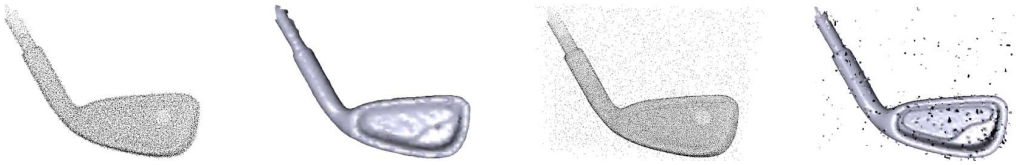

Fig. 12. Noise behavior of the method by Hoppe *et al.*: *Left-to-right*: points perturbed by Gaussian noise ($\sigma = 0.3\%$), reconstructed surface, 20% shot noise, reconstructed surface.

3) Random Sampling: We used a relatively large grid ($288 \times 408 \times 410$), such that the number of source points (100, 523) is comparable to the number of input points (100, 759). Then, keeping the grid resolution constant, we randomly subsampled the set of source points and performed reconstruction using only the sampled points; we used nearest-neighbor interpolation for grid assignment, and the parameters of the method were set as in the previous section. Fig. 9 shows the results. Note that the method yields very good results, even with as few as 10% of the source points.

C. Comparison to Other Methods

1) Mesh Smoothing: We compared our mesh-smoothing by a mass-spring system with curvature flow [47], [48], and with the noniterative, feature-preserving method of Jones *et al.* [49]. The results are shown in Fig. 10. The stopping time for the iterative methods (i.e., curvature flow and mass-spring system methods) was set to $t = 300$, whereas the parameters of the noniterative method were set to $\sigma_f = 2, \sigma_g = 10$ (to smooth large features), which yielded the best result. Note that the noniterative method preserves too many mesh details, whereas, at the other extreme, curvature flow smears out even large mesh features. Our proposed method seems to offer the best tradeoff between mesh smoothness and feature preservation. In addition, it can be efficiently implemented on GPU hardware, unlike the noniterative method.

2) Surface Reconstruction: We compared our surface reconstruction method to that of by Hoppe *et al.* [14], to the Power Crust algorithm by Amenta *et al.* [8], [9] and to the level set method of Zhao *et al.* [19], see Fig. 11. The time taken by our method for the model in Fig. 11 was 56 s on a grid with dimensions $450 \times 320 \times 206$; the reconstruction error was 0.06. It took 4 min by the method of Hoppe *et al.* to reconstruct the same model. Note that some holes are visible in the triangulated surface, since we increased the parameter controlling the sampling of the unknown surface as much as possible, in an attempt to reconstruct fine surface details. Although the reconstructed surface is smooth, fine surface details are lost. This method can tolerate Gaussian noise provided that each sample point has on average the same distance to its neighbors. However, the method does not tolerate shot noise, see Fig. 12.

The highest resolution of the reconstructed surfaces is obtained by methods which interpolate the data points, similar to the Power Crust algorithm, see Fig. 11. However, the time taken to reconstruct surfaces of large models (see Table I) within floating-point precision is two orders of magnitude larger than

Fig. 13. *Left*: noisy data set with 2 008 414 input points and 4000 outliers; reconstructed surface by: *center*—the proposed method (234 s), *right*—level set method (2 139 s).

TABLE II
STATISTICS, RECONSTRUCTION QUALITY AND TIMINGS OBTAINED USING THE LEVEL SET METHOD OF ZHAO *ET AL.* [19]. GRID RESOLUTIONS ARE THE SAME AS IN TABLE I. TIMINGS: **TOTAL 1**—MINIMAL SURFACE MODEL; **TOTAL 2**—CONVECTION MODEL WITH CURVATURE-BASED SMOOTHING

Model	No. Vertices	No. Triangles	Error (%)	Aggregation Time (s)	Tagging Time (s)	Convection Time (s)	Smoothing Time (s)	Minimal Surface Time (s)	Total 1	Total 2
Buddha	307,038	614,080	**0.09**	3.4	20.2	68.2	20.2	117.7	**210.4**	**113.1**
Armadillo	380,568	761,124	**0.07**	12.3	97.8	146.2	42.1	222.8	**480.5**	**301.3**
Dragon	391,216	782,432	**0.09**	6.2	44.3	77.9	31.3	151.5	**280.8**	**160.9**
Hand	186,006	372,036	**0.09**	5.4	38.7	83.0	32.4	120.2	**247.9**	**155.6**
Asian Dragon	213,256	426,276	**0.08**	6.9	57.5	102.3	36.1	132.6	**300.1**	**198.3**

that of the proposed method. Since this method interpolates the data points, it cannot cope with either types of noise which we considered.

One method which lends itself to an efficient GPU implementation is the level set method of Zhao *et al.* [19]. We performed a direct comparison of our method and this level set method, both implemented similarly on GPU hardware, as in Section IV. Statistics, reconstruction quality and timings for the latter method are shown in Table II. Instead of using the full minimal surface model, this method can be accelerated by first convecting the surface towards the data points and then using a small number of iterations (say 10) of curvature-based smoothing, to obtain the final (smooth) surface. The penultimate column of Table II) shows the total CPU time for the minimal surface model, whereas the last one shows the timings for the convection model with curvature-based smoothing. Note that even in the latter case, our method is about an order of magnitude faster than the level set method. Also, as shown in Fig. 13, this method does not tolerate shot noise, due to its reliance on the distance transform for aggregation.

One of the fastest techniques for surface reconstruction is the MPU method of Ohtake *et al.* [30]. Comparing the result from Table I on the Dragon data set with that from Table II in [30] one observes that our method is 2.3 times faster than the MPU method, at the same accuracy (8.0×10^{-4}). However, if larger accuracy is needed, their method may be more efficient. Nevertheless, the method assumes that accurate normal estimates are available.

TABLE III
GRID RESOLUTION VERSUS RECONSTRUCTION ERROR WITH/WITHOUT MESH SMOOTHING; RESULTS USING THE BUDDHA MODEL

Grid	Error (%) w/o Smoothing		Error bound, ε_r (%)	Total time (s) w/o Smoothing	
$67 \times 150 \times 67$	0.4	0.3	0.8	3.0	1.6
$108 \times 250 \times 108$	0.1	0.1	0.5	10.9	5.7
$190 \times 450 \times 190$	0.06	0.06	0.3	53.0	34.0
$272 \times 650 \times 273$	0.03	0.03	0.2	140.8	104.4
$354 \times 850 \times 355$	0.008	0.008	0.1	295.4	218.7

D. Error Analysis of the Framework

Assume that the grid resolution agrees with the (appropriate) sampling rate of the unknown surface to be reconstructed, i.e., each point of the data set is assigned to a distinct grid cell. In

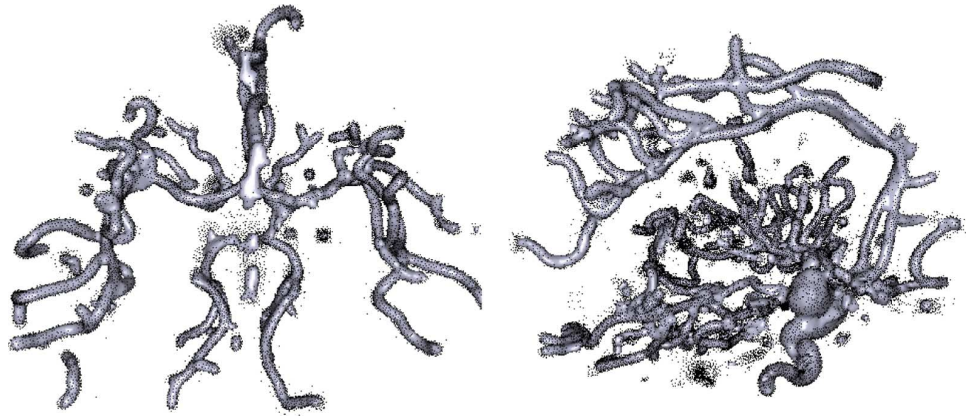

Fig. 14. Input from a particle system. The images show both the reconstructed surfaces and the input data points—given by particle positions.

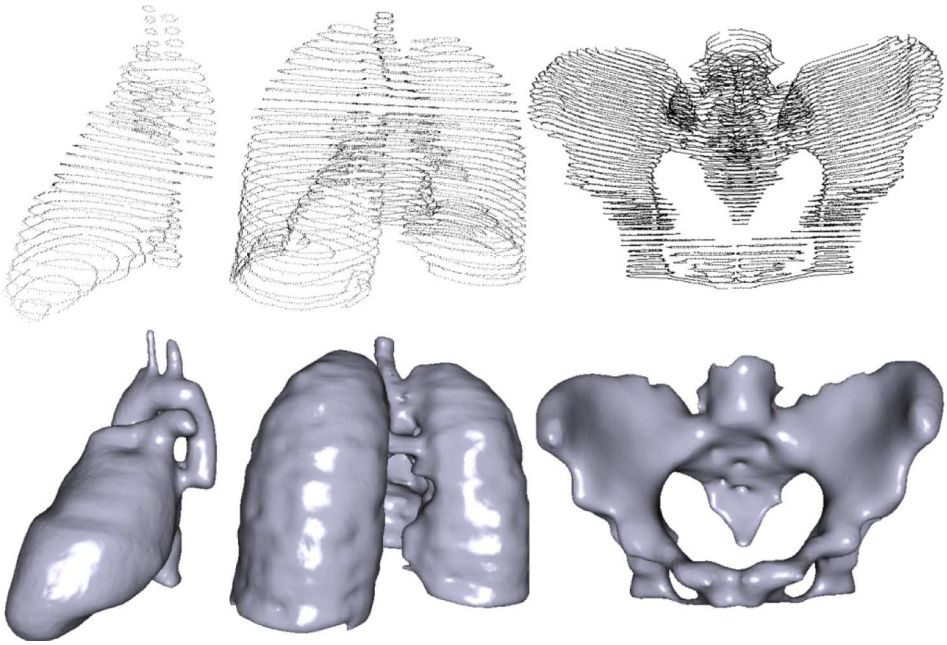

Fig. 15. Reconstruction from contours. *First row*: input contours; *second row*: reconstructed surfaces.

the small-cell-size limit, the CIC interpolation scheme becomes nearest neighbor interpolation. Also, we assume a clean input data set. After aggregation, a smooth scalar field emerges with regional maxima at the locations of the data points and ridges along straight segments connecting adjacent data points. After labeling, the exterior and interior layers bound the surface of the object closely, such that (i) grid points corresponding to input points in the boundary layer are enclosed between the interior and exterior layers, and (ii) the surface of the exterior/interior layer represents a manifold. Therefore, if $h_x = bx/gx$, $h_y = by/gy$, $h_z = bz/gz$ are the grid-cell sizes, bx, by, bz are the dimensions of the bounding box of the data points, and gx, gy, gz are the number of cells in the x, y, z directions, then a bound on the reconstruction error is given by the length of the diagonal of a grid cell, i.e., $\varepsilon = \sqrt{hx^2 + hy^2 + hz^2}$. Implicit surface interpolation by (3) with initial condition (4) cannot increase the reconstruction error since grid cells labeled as interior/exterior maintain their labels due to the similarity term. Moreover, interpolation using (3) yields a smooth field at boundary locations, which can only decrease the reconstruction error, though the error bound remains the same. After interpolation, the gradient field has correct orientation, without singular points, and, therefore, the reconstructed surface is *consistently oriented*. Also, when the grid resolution is large enough, any of the surfaces of the interior/exterior layers has the *same topology* as the unknown surface, and, therefore, the reconstructed surface will have the same topology.

In the presence of noise, surface features smaller than the noise amplitude in the data set can obviously not be recovered. However, as we showed in Section V-B, the method is noise tolerant, albeit the error bound increases up to $\varepsilon_r = \eta + \sqrt{hx^2 + hy^2 + hz^2}$, where η denotes the standard deviation of the noise. These error bounds remain the same even if the mapping of data points to nonempty grid cells is not one-to-one. The mass-spring system potentially increases the overall reconstruction error. Therefore, an essential requirement is that our mass-spring system preserves the features of the triangulated surface. We showed in Section V-C that this is indeed the case.

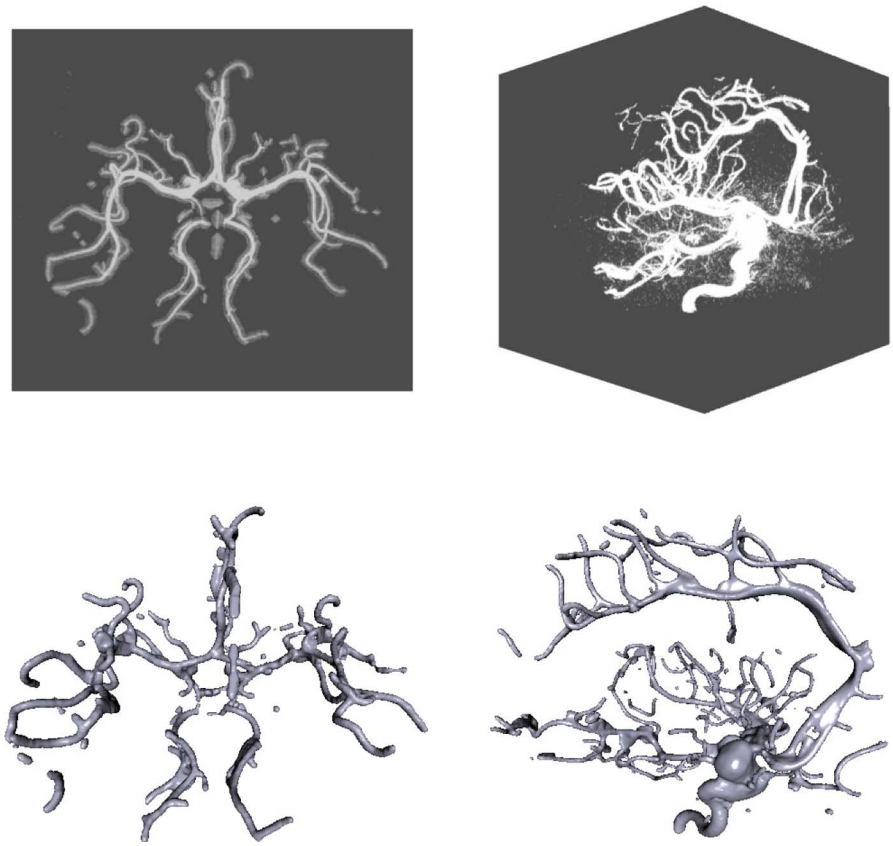

Fig. 16. Reconstruction from grayscale data (segmentation). *First row*: Maximum Intensity Projection (MIP) of the input data sets; *second row*: reconstructed surfaces.

E. Reconstruction Error

To verify our claims made in Section V-D, we studied the behavior of the reconstruction error (cf. Table I) when grid resolution increases, see Table III. As can be seen, the reconstruction error is always bounded by ε_r, even when the mass-spring system is used for mesh smoothing. Only at small grid resolutions the reconstruction error increases when mesh smoothing is applied, because our mesh smoother preserves small features of the triangulated surface.

The results shown in Table III were obtained using a CPU implementation, unlike those of Table I. The reason is the small amount of memory available on our current GPU hardware, which would not allow using grids larger than those of Table I.

F. Other Results

1) Particle Systems: To evaluate whether the proposed method can be used to triangulate surfaces sampled with particle systems, we used the particle system from [50] to provide the input data points; we have used two grayscale volumetric data sets, obtained by Magnetic Resonance Angiography (MRA). The results are shown in Fig. 14. Note that the method copes rather well with these noisy inputs, and yields in both cases smooth, approximating surfaces.

2) Contours: A Magnetic Resonance Imaging (MRI) scanner outputs parallel cross sections of the area under observation. Thus, one can extract polygonal contours in every cross section, either manually or using tools from image analysis [51]. Example results obtained by the proposed method using this kind of input data sets are shown in Fig. 15. Note that although these data sets are highly nonuniform, the method is able to output rather smooth surfaces.

3) Grayscale Volumetric Data: Another application is (semi-automatic) volumetric *segmentation*, based on the observation that instead of assigning the input points to the volumetric grid, one can use directly a grayscale volume as input. That is, we start by computing some contrast measure (e.g., gradient magnitude) within the input volume and by assigning heat sources proportional to this measure. Then, the method proceeds as usual with computing membrane potentials on this grid.

For the purpose of automatic segmentation one may use the bounding box as the initial surface, after the potentials have been computed. The resulting implicit surface defines the boundaries of the objects present in the input volume. This is in accordance with the very definition of image segmentation, i.e., the process by which an image is divided in its constituent parts. This implies that segmentation should produce a complete partitioning of the image such that object contours are closed and precisely localized. Note that since our method relies on an implicit-surface representation, the latter requirement is fulfilled.

The results of an experiment involving two sets of 3-D MRA volume data are shown in Fig. 16. The experimental setup was as follows. Since these data sets are likely to be noisy, we performed regularization of the input volumes (i.e., convolution

Fig. 17. Multiresolution. Grid resolutions (*left-to-right*): $600 \times 337 \times 402$, $400 \times 226 \times 269$, $200 \times 115 \times 136$.

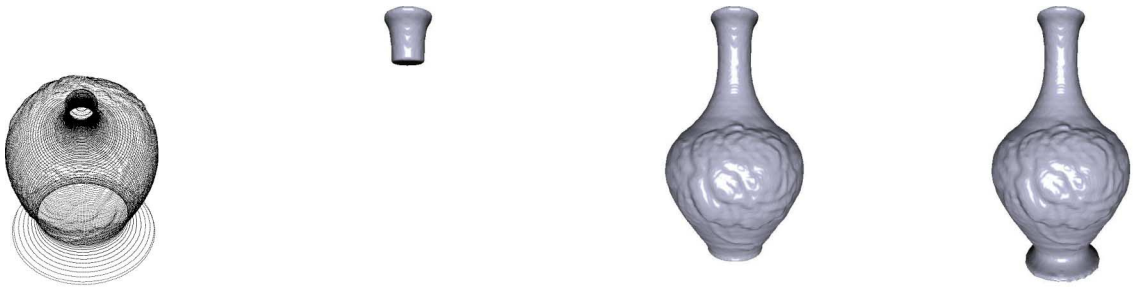

Fig. 18. Hole filling. *Left-to-right*: input points, reconstructions for $N_m = 40, 80, 100$.

with a Gaussian kernel of width $\sigma = 2.5$), prior to estimating the gradient magnitude. As can be seen from Fig. 16, most important structures were correctly recovered.

VI. Limitations

In closing, we list a number of (current) limitations of our method.

Surface features smaller than the grid size are not appropriately reconstructed, see Fig. 17. A possible solution would be to increase the grid resolution at the expense of larger computational time and memory requirements. Also, the method is not geometrically adaptive, but we are currently investigating an adaptive, multiresolution approach based on data-structures similar to octrees which can also be efficiently implemented on GPUs.

As is usual for methods that employ implicit surface representations, we assume that the surfaces to be reconstructed are closed, though the method does perform intrinsic hole filling by minimal surfaces. Fig. 18 shows that by increasing N_m (number of iterations used for computing membrane potentials), increasingly larger holes can be filled, at the expense of higher computational requirements. Note that there are no samples at the bottom of the vase object, as can be seen from the first image of Fig. 18.

A related problem is that if surface sheets come close to each other, e.g., see the knot surface in Fig. 4, and a large N_m is used, concavities and holes of the object may be filled.

VII. Conclusion

We have introduced a novel framework for surface reconstruction starting from unorganized point clouds without orientation information, and demonstrated its effectiveness in various experimental settings. The method can be used to efficiently reconstruct surfaces from clean as well as noisy data sets, and in our opinion, this represents an advantage over existing methods. The method can deliver multiresolution representations of the reconstructed surface, and can be used to perform reconstruction starting from particle systems, contours or even grayscale volumetric data leading to image segmentation.

Most constituent parts of the method have been implemented on GPU hardware. A problem which we encountered is the relatively small amount of video memory available on current graphics cards, which limits the sizes of the data sets which can be loaded in texture memory. Nevertheless, it is our belief that the method is flexible enough to be used and adapted to diverse tasks, ranging from accurate surface reconstruction on noise-free data sets to grayscale image segmentation.

References

[1] A. R. Kelly and E. Hancock, "A graph-spectral approach to shape-from-shading," *IEEE Trans. Image Process.*, vol. 13, no. 7, pp. 912–926, Jul. 2004.

[2] O. D. Faugeras and R. Keriven, "Variational principles, surface evolution, pdes, level set methods, and the stereo problem," *IEEE Trans. Image Process.*, vol. 7, no. 3, pp. 336–344, Mar. 1998.

[3] W.-L. Hwang, C.-S. Lu, and P.-C. Chung, "Shape from texture: Estimation of planar surface orientation through the ridge surfaces of continuous wavelet transform," *IEEE Trans. Image Process.*, vol. 7, no. 5, pp. 773–780, May 1998.

[4] D. Shin and T. Tjahjadi, "Local hull-based surface construction of volumetric data from silhouettes," *IEEE Trans. Image Process.*, vol. 17, no. 8, pp. 1251–1260, Aug. 2008.

[5] G. Atkinson and E. R. Hancock, "Recovery of surface orientation from diffuse polarization," *IEEE Trans. Image Process.*, vol. 15, no. 6, pp. 1653–1664, Jun. 2006.

[6] B. D. Rigling and R. L. Moses, "Three-dimensional surface reconstruction from multistatic sar images," *IEEE Trans. Image Process.*, vol. 14, no. 8, pp. 1159–1171, Aug. 2005.

[7] A. C. Jalba and J. B. T. M. Roerdink, B. S. Santos, T. Ertl, and K. Joy, Eds., "Efficient surface reconstruction from noisy data using regularized membrane potentials," in *Proc. Eurographics/IEEE VGTC Symp. Visualization*, Lisbon, Portugal, May 8–10, 2006, pp. 83–90, The Eurographics Association.

[8] N. Amenta, M. Bern, and D. Eppstein, "The crust and the β-skeleton: Combinatorial curve reconstruction," *Graph. Model Image Process.*, vol. 60, no. 2, pp. 125–135, 1998.

[9] N. Amenta, M. Bern, and M. Kamvysselis, "A new Voronoi-based surface reconstruction algorithm," in *Proc. SIGGRAPH*, 1998, pp. 415–421.

[10] J. D. Boissonnat, "Geometric structures for three-dimensional shape reconstruction," *ACM Trans. Graph.*, vol. 3, no. 4, pp. 266–289, 1984.
[11] H. Edelsbrunner and E. P. Mücke, "Three-dimensional alpha shapes," *ACM Trans. Graph.*, vol. 13, no. 1, pp. 43–72, 1994.
[12] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Trans. Vis. Comput. Graph.*, vol. 5, no. 4, pp. 349–359, Apr. 1999.
[13] J. D. Boissonnat and F. Cazals, "Smooth surface reconstruction via natural neighbor interpolation of distance functions," in *Proc. 16th Annu. Symp. Computational Geometry*, 2000, pp. 223–232.
[14] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," in *Proc. SIGGRAPH*, 1992, pp. 71–78.
[15] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. SIGGRAPH*, 1996, pp. 303–312.
[16] C. T. Lim, G. M. Turkiyyah, M. A. Ganter, and D. W. Storti, "Implicit reconstruction of solids from cloud point sets," in *Proc. 3rd ACM Symp. Solid Modeling and Applications*, 1995, pp. 393–402.
[17] C. L. Bajaj, F. Bernardini, and G. Xu, "Automatic reconstruction of surfaces and scalar fields from 3D scans," *Comput. Graph.*, vol. 29, pp. 109–118, 1995.
[18] C. K. Tang and G. Medioni, "Inference of integrated surface, curve, and junction descriptions from sparse 3-D data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, pp. 1206–1223, 1998.
[19] H. Zhao, S. Osher, and R. Fedkiw, "Fast surface reconstruction using the level set method," in *Proc. IEEE Workshop on Variational and Level Set Methods in Computer Vision*, 2001, pp. 194–202.
[20] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *J. Comput. Phys.*, vol. 79, pp. 12–49, 1988.
[21] U. Clarenz, U. Diewald, and M. Rumpf, "Processing textured surfaces via anisotropic geometric diffusion," *IEEE Trans. Image Process.*, vol. 13, no. 2, pp. 248–261, Feb. 2004.
[22] A. I. El-Fallah and G. E. Ford, "Mean curvature evolution and surface area scaling in image filtering," *IEEE Trans. Image Process.*, vol. 6, no. 5, pp. 750–753, May 1997.
[23] O. M. Lysaker, S. Osher, and X.-C. Tai, "Noise removal using smoothed normals and surface fitting," *IEEE Trans. Image Process.*, vol. 13, no. 10, pp. 1345–1357, Oct. 2004.
[24] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, "Reconstruction and representation of 3D objects with radial basis functions," in *Proc. SIGGRAPH*, 2001, pp. 67–76.
[25] B. S. Morse, T. S. Yoo, P. Rheingans, D. T. Chen, and K. R. Subramanian, "Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions," in *Proc. Shape Modeling International*, 2001, pp. 89–98.
[26] N. Kojekine, V. Savchenko, and I. Hagiwara, "Surface reconstruction based on compactly supported radial basis functions," in *Geometric Modeling: Techniques, Applications, Systems and Tools*. Boston, MA: Kluwer, 2004, pp. 218–231.
[27] H. Q. Dinh, G. Turk, and G. Slabaugh, "Reconstructing surfaces by volumetric regularization using radial basis functions," *IEEE Trans. Pattern Anal. Machine Intell.*, pp. 1358–1371, 2002.
[28] G. Turk and J. O'Brien, Variational Implicit Surfaces, Georgia Inst. Technol., Atlanta, Tech. Rep., 1999.
[29] Y. Ohtake, A. Belyaev, and H. Seidel, "Multi-scale approach to 3D scattered data interpolation with compactly supported basis functions," in *Proc. Shape Modeling International*, 2003, pp. 153–164.
[30] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H. Seidel, "Multi-level partition of unity implicits," in *Proc. SIGGRAPH*, 2003, pp. 463–470.
[31] J. Carr, R. Beatson, B. McCallum, W. Fright, T. McLennan, and T. Mitchell, "Smooth surface reconstruction from noisy range data," in *Proc. Graphite*, 2003, pp. 119–126.
[32] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proc. SIGGRAPH*, 1987, pp. 1631–169.
[33] P. Ning and J. Bloomenthal, "An evaluation of implicit surface tilers," *IEEE Comput. Graph. Appl.*, vol. 13, pp. 33–41, 1993.
[34] J. Bloomenthal, *An Implicit Surface Polygonizer*. San Diego, CA: Academic, 1994, pp. 324–349.
[35] J. P. Christiansen, "Numerical simulation of hydromechanics by the method of point vortices," *J. Comput. Phys.*, vol. 13, pp. 363–379, 1973.
[36] D. Scharstein and R. Szeliski, "Stereo matching with nonlinear diffusion," *Int. J. Comput. Vis.*, vol. 28, pp. 155–174, 1998.
[37] D. Terzopoulos, "Regularisation of inverse visual problems involving discontinuites," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, pp. 413–424, 1986.
[38] C. Xu and J. Prince, "Snakes, shapes, and gradient vector flow," *IEEE Trans. Image Process.*, vol. 7, no. 3, pp. 359–369, Mar. 1998.
[39] C. Y. Kao, S. Osher, and J. Qian, "Lax-Friedrichs sweeping scheme for static Hamilton-Jacobi equations," *J. Comput. Phys.*, vol. 196, pp. 367–391, 2004.
[40] M. Vasilescu and D. Terzopoulos, "Adaptive meshes and shells: Irregular triangulation, discontinuities, and hierarchical subdivision," in *Proc. CVPR*, 1992, pp. 829–832.
[41] G. Taubin, "Estimating the tensor of curvature of a surface from a polyhedral approximation," in *Proc. ICCV*, 1995, pp. 902–907, IEEE Computer Society.
[42] H. P. Moreton and C. H. Séquin, "Functional optimization for fair surface design," in *Proc. SIGGRAPH*, New York, NY, 1992, pp. 167–176.
[43] L. Verlet, "Computer experiments on classical fluids I. Thermodynamical properties of Lennard-Jones molecules," *Phys. Rev.*, vol. 159, pp. 98–103, 1967.
[44] M. Harris, G. Coombe, T. Scheuermann, and A. Lastra, "Physically-based visual simulation on graphics hardware," presented at the SIGGRAPH/Eurographics Workshop on Graphics Hardware, 2002.
[45] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid," in *Proc. SIGGRAPH*, 2003, pp. 917–924.
[46] V. Pascucci, "Isosurface computation made simple," in *Proc. VisSym*, 2004, pp. 293–300.
[47] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, , H.-C. Hege and K. Polthier, Eds., "Discrete differential-geometry operators for triangulated 2-manifolds," in *Visualization and Mathematics III*. Heidelberg, Germany: Springer-Verlag, 2003, pp. 35–57.
[48] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proc. SIGGRAPH*, New York, 1999, pp. 317–324.
[49] T. R. Jones, F. Durand, and M. Desbrun, "Non-iterative, feature-preserving mesh smoothing," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 943–949, 2003.
[50] A. C. Jalba, M. H. F. Wilkinson, and J. B. T. M. Roerdink, "CPM: A deformable model for shape recovery and segmentation based on charged particles," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, pp. 1320–1335, 2004.
[51] J. D. Boissonnat, "Shape reconstruction from planar cross-sections," *Comput. Vis. Graph. Image Process.*, vol. 44, pp. 1–29, 1988.

Andrei C. Jalba received the B.Sc. (1998) and M.Sc. (1999) degrees in applied electronics and information engineering from the "Politehnica" University of Bucharest, Romania, and the Ph.D. degree from the Institute for Mathematics and Computing Science, University of Groningen, Germany, in 2004.

His research interests include computer vision, pattern recognition, image processing, and parallel computing.

Jos B. T. M. Roerdink (SM'95) received the M.Sc. degree (1979) in theoretical physics from the University of Nijmegen, The Netherlands, and the Ph.D. degree (1983) from the University of Utrecht.

Following his Ph.D. and a two-year position (1983–1985) as a Postdoctoral Fellow at the University of California, San Diego, both in the area of stochastic processes, he joined the Centre for Mathematics and Computer Science, Amsterdam, The Netherlands. He worked there from 1986–1992 on image processing and tomographic reconstruction. He was appointed to Associate Professor (1992) and Full Professor (2003), respectively, at the Institute for Mathematics and Computing Science, University of Groningen, where he currently holds a Chair in Scientific Visualization and Computer Graphics. His research interests include biomedical visualization, neuroimaging, and bioinformatics.